Design Document

inHouse

Austin Bolingbroke, Cody Davis, Danny Harding, Melanie Lambson, Scott McKenzie, Chase Robertson, Hojune Yoo

Contents

Contents 2

Purpose of Document 3

Product Purpose 3

Overview 3

iOS App Design 3
- Overview 3
- Details 4
  - App Flow 4
- Detail/Profile Pages 5
  - Model 5
  - Server Communicator 5
  - Payment 6

Website Design 6
- Overview 6
- Details 6
  - Hosting and Environment 6
  - Web Pages 7
  - Server Communication 8

Backend Design 8
- Overview 8
- Details 8
  - Database 8
  - ORM 9
  - Token Authentication 9
  - RESTful API 10
  - HTTP Request / Response 10

Extra Information? 10

Purpose of Document

This Design Document provides an outline for the design of our product that will fulfill the technical requirements stated in the requirements document. Within this document, we explicitly state how our product will satisfy our previously defined requirements. We didn't originally label the requirements in the requirements doc, and so did not reference the requirement numbers in this document. Once these numbers are added, we will add the references here.

Product Purpose

Our team is seeking to fill a hole within the restaurant industry by allowing customers to take control of their restaurant experience. Many restaurant customers want to enjoy a restaurant at their own pace, not that of the waiter or waitress. inHouse will allow these customers to enjoy the restaurant experience on their own terms. We also aimed to offer a versatile solution that is easy to implement for small restaurants and that would require a minimum investment (no need to install costly hardware or tablets in the restaurant).

Overview

inHouse will consist of three separate pieces. First, an iOS app for customers to interact with a restaurant when they visit a restaurant to be served. We will also have a website for participating restaurants that allows the restaurant to manage what the customer can see on the app. The last piece is a backend consisting of a server and database that connects the two clients (app and website) and keeps the information up to date.

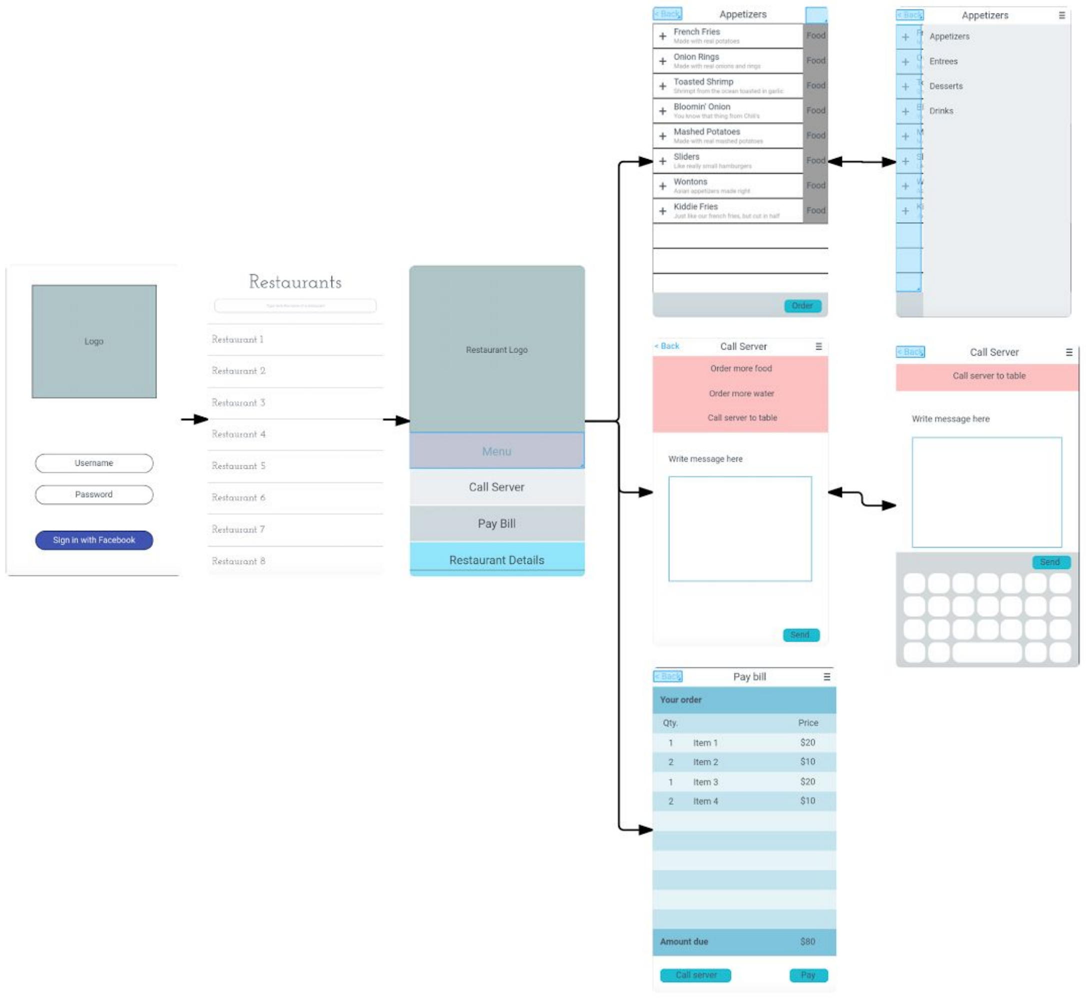
iOS App Design

Overview

The iOS app will have a small set of views connected in a very simple flow. Once connected through a signup page, the user will access the main page consisting of a list of participating restaurants. Upon selecting a restaurant, the user is taken to the restaurant page. From here the user can access functionality specific to the restaurant, such as the ability to call a server or to pay a bill, as well as ordering food. There will also be a profile page for the user, as well as a dedicated payment page to view the receipt and pay for the food. As is the standard with iOS development, we will follow the MVC (Model-View-Controller) pattern to separate the display and the data management in our app.

Details

App Flow



The preceding diagram demonstrates the flow of each view in the app. The restaurant list screen is the home page, which a user must be logged in to see. Selecting a restaurant leads to the restaurant page. This page is a simplistic menu of actions that are used to interact with the restaurant in some way. The menu page displays all items on the menu by category. By selecting a new category on the menu page, the list of food is updated to display only that food which is in the selected category. Any menu item may be added to the current order with the add to order button (+), and the order can be placed with the “Order” button.

The “Call Server” option on the restaurant page opens a page that will allow communication with the restaurant. You can ask for more water or send a custom message. Selecting the “order more food” option will take the user back to the menu view.

The “Pay Bill” option on the restaurant page displays the payment page. This is a static page and will display each ordered food item, its cost, and the total amount to pay. There will be a button which opens a modal to accept credit card information and complete the transaction, as well as a button to call the server with any questions. While the user is paying, they will also be prompted to provide a tip.

Detail/Profile Pages

Our app will include a user profile page as well as a restaurant details page. These pages will display information specific to the entity they represent. The user profile page will display the user’s information, such as name, phone number, email, and picture. It will also show app specific data such as last visited restaurant, time of last visit, etc.

The restaurant details page will show the restaurant’s information such as name, type of restaurant, address, phone number, logo, and image of the building. Future versions could include data such as number of inHouse users who have visited the restaurant, inHouse ratings and other data that is specific to inHouse for that restaurant.

Model

The model will need to consist of various classes to manage the data retrieved from the server. The classes needed for viewing restaurant information are Restaurant, Menu, and MenuItem. We will need an Order class as well which will contain MenuItems and the combined price.

Apart from restaurant related model classes, we will have a User class that is responsible for keeping track of the current login state and who is actually logged on the device. This will give most users the ability to use the app without having to login each time the app is opened.

Server Communicator

The purpose of the Server Communicator is to process requests to be sent to the server and ensure that proper information is received in return. It will process any and all HTTP requests to the server and manage the responses. This decouples the process of interacting with the server from the view-controllers as well as the model.

Payment

We are investigating the different options for making payments to the restaurant. We will use a third party integration (most likely Stripe or Braintree) to accept payment and pass it on to the restaurant. We have not decide whether the app will make the payment request directly or if it will go through the server. This design decision will be made as we come closer to that point.

Website Design

Overview

The purpose of the website is to provide a web portal for a restaurant to sign up, log in, and fulfill orders. The website will consist of a simple informational page with links to login and register. Once logged in, a restaurant will be able to create and edit a menu, see and fulfill current orders, and assign customers to a table. The website will communicate with the server to keep the data up to date and accessible to the iOS app as well as any other app that may need the data in the future.

Details

Hosting and Environment

While the website could be hosted on almost any server, we have chosen to host it on an Amazon EC2 instance. The free tier should provide more than enough resources for our web app and in the event that we wanted to expand, it is easily scalable. The instance will serve the basic pages mentioned below.

The pages will be HTML with AngularJS to implement some of the functionality. Using Angular, we will be able to reuse sections of code on the different pages instead of adding repeated elements to each individual page. We will also be able to code logic for displaying and handling the data using Javascript.

User Authentication

To secure individual restaurant data, we will use token authentication. A restaurant will be required to login before any data is shown to the user. This will prevent anonymous users from viewing and editing a restaurant's data. A user will be presented with a login screen which allows them to enter in their credentials. Once their credentials are entered and validated by the server, a token is created by the server and sent back. From then on, every communication sent to the server includes this token. As long as the token is valid, the server will return the data. Otherwise, the server denies the data request.

Web Pages

Landing Page: This page will have basic information about the product. This page will also have a link to login for current clients, and a link to register for potential clients.

Login Page: The login page will have simple text box field for a customer to login.

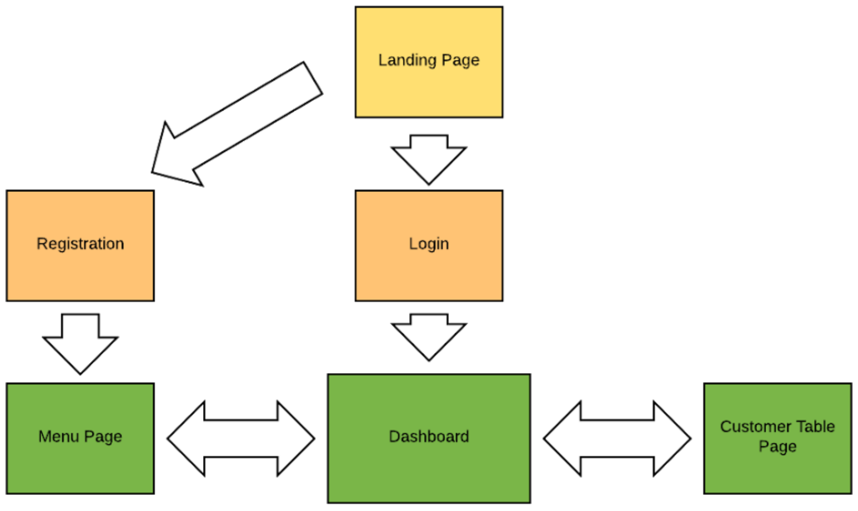
Registration Page: The registration page will collect all the necessary data to create a new restaurant in the database.

Dashboard: The dashboard will be the main page of the web app. This is where a restaurant will be taken after they login. The dashboard will have awaiting orders so that a restaurant can know what needs to be cooked and served. Once an order has been served, a restaurant will be able to mark the order as fulfilled. This page will also have links to the Menu page and the Customer Table page.

Menu Page: On the menu page, a restaurant can create the menu that will be available to customers on the iOS app. They will be able to add a menu item that has a title, image, price, and description. They can also add options to the menu item such as sauce, sides, cooking time, or other variations a customer needs to make. They will also be able to edit and delete current menu items. This is the page a restaurant will be taken to after registration.

Customer Table Page: On this page, a restaurant will be able to assign a customer to a table. Once the customer has been assigned to a table, it will generate a code to give to the customer. The code can be entered by the customer into the app to connect that customer with a particular table in the restaurant. This page will also allow the restaurant to specify what tables exist in the restaurant.

Basic Page Flow:



Server Communication

To show the current order status and other data to the user, the website will need to communicate with the server. The website will use communicate through AJAX requests to the server. The server has a REST service which we can communicate through. The website will not use a database of its own since the server will be storing all the information. Through the website, a user can view and edit the information stored on the server. To keep the information about current orders and customers current, the website will regularly poll the server for the latest data.

Backend Design

Since our app will need to handle multiple restaurants, servers and customers, it is necessary to include a backend to store all of the information. Throughout the design of the backend, we focused on scalability to make it as robust as possible.

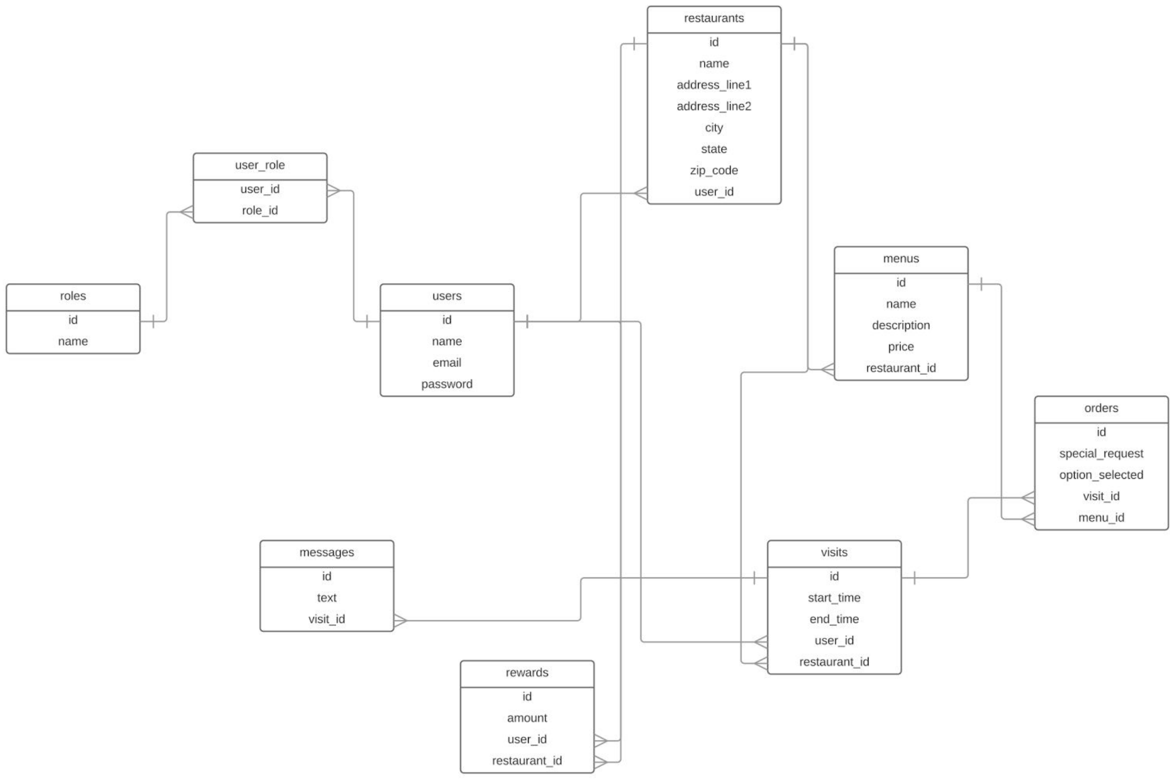
Overview

Our backend system runs on Linux CentOS 6.5, PHP7, MySQL5.6, in an Apache2 environment. The backend system will help client side application to retrieve correct data with minimum api calls and store data upon requests from client side applications. Backend system runs on Laravel framework with Dingo API to provide RESTful API based on token authentication.

Details

Database

Currently we have 11 tables in our relational database. We have focused on scalability of our database and it can handle different restaurants regardless of their menus and menu options. The User, Restaurant, Menu, Order, and Visit tables are 5 of the most important databases we have. Every table schema will be managed through migration files. Every update on database tables and columns will be recorded on migration files. Also dummy seeds will be added into the database tables through seed files for testing purposes. The following UML diagram illustrates the relations between the database tables.



ORM

Each database table except pivot tables has its own model and it is possible to retrieve data through ORM. Since our database is relational and using ORM, it is very easy to retrieve other related models. For example to retrieve restaurant with menus, we can just simply write a one line code (Restaurant::with('menuCategories.menus');) Using ORM gives us the advantage of writing and managing much cleaner code.

Token Authentication

To secure the connection between server and client we are going to use token authentication. Client application and server application will share a token during the login process and token will expires after some amount of time of idleness.

RESTful API

For each database model, the server will provide a resource controller which will include RESTful CRUD methods. Some of the more delicate models may not provide update or delete methods, and will require special roles to grant access. These will be determined as we understand more of what is needed.

HTTP Request / Response

The server will return only JSON objects as a response, while requests from the client applications can be JSON, form, query string, etc. Every possible http request will be documented through swagger-ui which will be accessible through /api/documentation.